Software Engineering Institute

Workshop on Model-Driven Architecture and Program Generation

Grace A. Lewis
B. Craig Meyers
Kurt Wallnau

August 2006

TECHNICAL NOTE
CMU/SEI-2006-TN-031

Predictable Assembly from Certifiable Components (PACC)
Integration of Software-Intensive Systems (ISIS)

Carnegie Mellon

Software Engineering Institute

Workshop on Model-Driven Architecture and Program Generation

Grace A. Lewis
B. Craig Meyers
Kurt Wallnau

August 2006

TECHNICAL NOTE
CMU/SEI-2006-TN-031

Predictable Assembly from Certifiable Components (PACC)
Integration of Software-Intensive Systems (ISIS)

Carnegie Mellon

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

List of Figures

List of Tables

Acknowledgements

We thank the invited attendees for their interest and participation in this work: Michael Jungmann, Pat Kohli, Douglas C. Schmidt, Fred Waskiewicz, and Ben Watson.

We also thank Software Engineering Institute technical staff members David Fisher, James Ivers, and Reed Little for their participation. We offer special thanks also to Laura Huber for her assistance in organizing the event.

Abstract

This technical note summarizes the results of a workshop held on June 2, 2006, at the Software Engineering Institute in Pittsburgh, Pennsylvania (USA). The workshop explored business and technical aspects of program generation in the context of the Object Management Group's model-driven architecture development approach. The workshop was structured around consideration of the perspectives of five different communities: standards body, vendor, acquisition, development, and research. This note recapitulates these individual perspectives and highlights important themes.

1 Introduction

This technical note describes the results of a workshop focused on model-driven architecture (MDA) for automated program generation. The term MDA refers to the approach and standards developed by the Object Management Group (OMG).[1] The workshop was held at the Carnegie Mellon® Software Engineering Institute (SEI) on June 2, 2006 in Pittsburgh, Pennsylvania (USA). The purpose of the workshop was to have an open and informal conversation about substantive business and technical aspects of program generation in the context of MDA.

MDA is a well-known approach to the development and maintenance of systems. MDA is produced and maintained by OMG, an open-membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. The goal of MDA is one that is often sought: to separate business and application logic from its underlying execution platform technology so that (1) changes in the underlying platform do not affect existing applications and (2) business logic can evolve independently from the underlying technology [Lewis 04, OMG 03].

A tool that implements the MDA concept would allow developers to produce models of the application and business logic and generate code for a target platform by means of transformations. Instead of writing platform-specific code in some high-level language, developers focus on developing models that are specific to the application domain but independent of the platform. In this way, MDA raises the level of abstraction in software development.

MDA has been endorsed by various entities of the U.S. Department of Defense (DoD). For example, in a recent Defense Science Board report on the Joint Integrated Fire Support Systems (JIFSS), a large distributed system, it was stated that JIFSS

> ... should employ the Model-Driven Architecture (MDA) development approach for designing the JIFSS architecture and in implementing its component systems. The MDA approach ensures adherence to standards across the components and has been shown to substantially reduce costs in the development of large-scale systems-of-systems [ATL 04].

Statements such as the one above are predicated on the expected benefits of using MDA on large systems. Will MDA satisfy these expectations? Which elements of MDA are robust today, and which will require further development? How will MDA change DoD

[1] For more information about the OMG, go to http://www.omg.org.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

procurement practice? These are important (if only representative) questions to ask, but they are difficult to answer—for any new technology, but more so because of the complexity of MDA itself. To make progress toward finding answers, we necessarily restricted the scope of our inquiry in our workshop. For this workshop we selected **program generation** (or **model transformation** in MDA terms) as our scope.

We elicited **perspectives** on the state of MDA technology in our workshop, specifically those views relating to automated model transformation and automated program generation. We sought the perspectives of five particular MDA-related communities:

1. standards body
2. vendor
3. acquisition
4. development
5. research

Each of these communities has different desires and expectations for what MDA can provide. Bringing them together in a small forum was expected to provide interactions that not only focused on the individual perspectives but also afforded the opportunity for integration of these perspectives.

The workshop was organized as a small, invitation-only event. Each of the invited participants is expert in the OMG MDA development approach and could plausibly play the role of perspective representative.

Several important observations emerged during the course of the workshop:

- MDA is best thought of as a software engineering approach that is supported by a set of *du jour* standards. The benefits of MDA will emanate primarily from the approach and secondarily from the technical details of the standards themselves.
- MDA tools today are exhibiting increasing capabilities to generate code from models. On the other hand, support for model-based analysis is lagging, at least in part because it introduces challenges beyond the state-of-the-research.
- The use of MDA in DoD acquisition processes has been motivated primarily by anticipated technology benefits. How the acquisition of models of systems instead of their implementations will change acquisition remains to be determined.

These observations are elaborated in the main text of this note. The remainder of this note is organized as follows:

- Section 2 provides some background on MDA.
- Section 3 discusses the workshop approach, including purpose, scope, attendees, and conduct of the workshop.

- Section 4 presents a discussion of the role-based perspectives represented by attendees. The perspectives, based on the five communities mentioned above, were standards body, vendor, acquirer, developer, and researcher.
- Section 5 discusses a thematic perspective of material discussed in the workshop. The themes arising from that discussion deal with
  - understanding MDA
  - the role of MDA in the software development life cycle
  - transformations and generator technologies
  - present state of MDA capabilities
  - nonfunctional attributes
  - validation
  - domain-specific languages
- Section 6 contains a brief summary of the major findings of the workshop.

2 Background

MDA is a broad conceptual framework that describes an overall approach to software development. In general, in tools that implement the MDA concept, a platform-independent model (PIM) that contains only business and application logic is developed using the tool's modeling component. The code generation component is a series of transformation rules that map elements in the PIM to elements in a platform-specific model (PSM) that contains details that are specific to the target platform.

In this way, all code that depends on the middleware, for example, will be generated and not written by hand as is usually the case today. Code is considered a form of the PSM, and a PIM can go through several levels of transformation before becoming code. This model transformation process is illustrated in Figure 1.

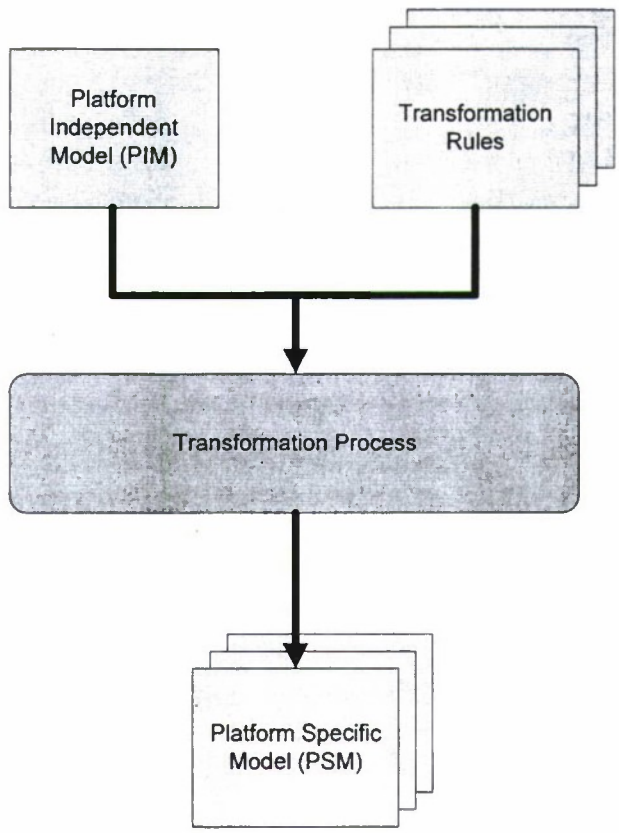

Figure 1: MDA Model Transformation Process

Some of the expected benefits of MDA, as identified by OMG, include:

- reduced cost throughout the application life cycle
- reduced development time for new applications
- improved application quality
- increased return on technology investments
- rapid inclusion of emerging technology benefits into their existing systems [OMG 06]

The OMG has developed the fundamental concepts of MDA and has working groups defining standards that are necessary to realize those concepts in practice. While the MDA concept is a vendor- and technology-neutral approach, MDA is compatible with

- established OMG standards such as
  - CORBA (Common Object Request Broker Architecture)
  - UML (Unified Modeling Language)
  - MOF (MetaObject Facility)
  - XMI (XML Metadata Interchange)[2]
- industry standards such as Web Services
- component frameworks such as
  - Sun Microsystems's J2EE (Java 2 Platform, Enterprise Edition)
  - Microsoft's .NET [OMG 03]

[2] XMI stands for XML Model Interchange. It is an OMG specification used for model exchange between most tools that have an XML model feature.

3 Workshop Approach

3.1 Attendees

The attendees, invited by the SEI, represented a broad range of depth and experience in dealing with various aspects related to MDA development approach. Each attendee represented a particular area of expertise and represented one of the following communities:

- standards body (Mr. Fred Waskiewicz, OMG)
- vendor (Mr. Michael Jungmann, Interactive Objects)
- acquisition (Mr. Pat Kohli, Naval Air Systems Command—NAVAIR)
- development (Mr. Ben Watson, Lockheed-Martin Corporation)
- research (Dr. Douglas C. Schmidt, Vanderbilt University)

In addition to the authors of this report, several other SEI technical staff members participated in the workshop.

3.2 Conduct of the Workshop

The agenda for the workshop is summarized below:

1. **Introduction**
   This segment introduced the workshop and set the expectations of the attendees.
2. **Overview of MDA**
   A brief overview of MDA was given to set a common theme.
3. **SEI Interest in MDA**
   A brief discussion of why the SEI was interested in MDA was presented.
4. **Presentation of Role-Based Perspectives**
   Assuming the five roles, attendees were asked to present their perspectives on MDA. The questions that each perspective was expected to address are presented in Table 1. It should be noted that having various perspectives presented was a significant factor in the success of the workshop.
5. **Development of Thematic Perspectives**
   The perspectives of the roles represented by the attendees were integrated to provide thematic perspectives.
6. **Summary**
   The SEI and attendees presented summaries of the workshop content.

Table 1: MDA Preparation Questions to be Addressed by Each Role-Based Perspective

Community	Role-Based Perspective	Questions
Standards body	Standards body	• Where is MDA heading? What are the next steps? • In your opinion, what are the MDA aspects that vendors need to implement in their tools to get broader adoption? • What is the status of standards/proposals/recommendations (such as QVT) that will make tool and model interoperability possible? • How broadly accepted is MDA? Are there a lot of people using it, and if so what communities do they represent?
Vendor	Vendor	• What MDA capabilities or extensions beyond MDA require further research? • What aspects of MDA are most critical in "closing the deal"? • With which aspects of MDA do your users need the most assistance to effectively use your tool?
Acquisition	Acquirer	• What role can MDA play in an acquisition strategy? It will be useful to discuss what the acquirer most expects to gain from the use of MDA. • When an acquisition organization says that it wants to acquire models, what does that mean exactly?
Development	Developer	• What are the most effective aspects of MDA in practice? • What aspects of MDA are not as effective as you had hoped or expected? • What aspects not currently addressed by MDA do you think are needed or would be most desirable? • How do developers deal with nonfunctional aspects, like performance or deadlines? • What are your experiences with MDA code generator tools?
Research	Researcher	• How does MDA fit in the context of past and current research in specification, analysis, and program generation? It is "ok" to address both natural strengths and limitations inherent to the MDA approach. • Can MDA be a conduit for commercial exploitation of software engineering technology research? • What can tool vendors and the OMG do to facilitate this outcome?

4 Role-Based Perspectives

This section describes the perspectives presented by the attendees based on their roles and summarizes the material they presented. The summaries that follow should not be taken to represent positions of the participants but rather the positions of the roles they were asked to represent. The authors have taken some liberties in emphasizing some aspects of presentations at the expense of others.

4.1 Standards Body Role-Based Perspective

The primary standards organization involved in MDA work is the OMG. Although the OMG has an architecture steering board, its standards are member-defined. This emphasis on user preferences results in a varied set of standards that address the needs and concerns of the organization's members. The work of OMG is broadly accepted and used in enterprise systems (e.g., The Walt Disney Company, Deutsche Bank AG, transportation organizations, and healthcare organizations), as well as by the real-time and embedded communities (notably the DoD).

The following definition was proposed initially

> **model-driven architecture:** a model-based, standards-driven, and tool-supported software engineering approach to application and system development

The goal of MDA is achieved through abstraction, automation, and the use of open (industry-based) standards. An implementation of the MDA development approach is realized through the use of a PIM and various possible PSMs, as was shown in Figure 1 on page 4. Transforming a PIM to a PSM accounts for the platform-specific characteristics, including distribution, middleware, and operating system dependencies. Thus, the same PIM can be applied to a single-process, single-processor model or to a multithreaded, multiprocessor model.

In OMG's MDA, models are represented using UML. Some elements of the UML have well-defined execution semantics[3] (e.g., state machine specifications and sequence diagrams). The UML Object Constraint Language (OCL) permits specification of pre- and post-conditions, invariants, and other conditions on various UML model elements.

[3] There is invariably some controversy about how rigorous this definition is. For the purpose of this report, we assume that the current definition is sufficiently rigorous for some but not all purposes. There is ongoing work at the OMG to provide a fully formal (execution) semantics for a subset of the UML. This difficult undertaking will likely never be extended to the full suite of UML notations.

Other MDA-related standards include:

- Knowledge Discovery Metamodel that provides a common repository facilitating discovery and exchange of data
- Abstract Syntax Tree Metamodel that is used to specify the structure of application code
- Business Process Management that includes process definition, modeling and representation of production rules
- metadata management that includes
  - a framework and services to enable development and interoperability of models and metadata systems (an MOF)
  - Query, View, Transformation (QVT)
  - Information Management Metamodel based on UML profiles

The development of standards by OMG continues, along with that of specifications related to MDA tool interoperability. Since OMG standards are defined by its members, the MDA portfolio of standards will continue to change to reflect the interests of the membership. On the one hand, this method can enable the standards to adapt to changing needs; on the other hand, it can also result in an unwieldy portfolio of standards.

4.2 Vendor Role-Based Perspective

For an MDA-based tool vendor, it is important to understand the problem that MDA and its users are trying to solve. It is common to see different interpretations, because perceptions of these problems have evolved over time. Is the problem more than just code generation? It probably is, as can be seen from this unordered list of potential answers:

- Achieve platform independence and interoperability.
- Simplify and streamline system development.
- Protect investments by turning knowledge into accessible and reusable assets.
- Help the business people understand what the system does.
- Industrialize the software development process by separating roles and supporting workflow.
- Plan, create, and maintain holistic enterprise information technology (IT) landscapes.

Given this varied list of answers, it is no surprise that adding a feature to a product seems like **research** for a vendor. The feature corresponds either to an aspect of MDA that is not fully-developed or to user expectations that stem from the evolving answer to the problem that MDA should solve.

A common question from potential customers to vendors is "What can your product do for me?" This question is hard to answer, because an appropriate response will always depend on the context. It follows, then, that a key aspect for closing a deal is working with the customer to identify the right problem that the product will solve. As part of that process, a vendor

needs to identify MDA as a solution for a seemingly unsolvable problem. Other generic critical factors for the vendor are

- efficient customization of a solution to customer's needs
- efficient support of target technology
- efficient integration with development environment
- minimization of risk

Once the product is acquired, customers require more assistance at first, probably due to the immaturity of the technology and the need to overcome a steep learning curve. A main area of vendor assistance is the development of metamodels, transformations, and chains of transformations.

From a vendor perspective, there are some interrelated potential areas of research associated with MDA that would help develop a better product. Some of these are

- domain-specific modeling and scalable modeling environments
- end-to-end software development life-cycle support and integrated workflow management as a form of development process control
- round-trip (generation and transformation) support that goes beyond trivial mappings or incremental transformations and automated generation of artifacts other than code and infrastructure[4]
- clearer relationships between MDA and software reuse
- specification, notation, and integration of metamodels
- industry-specific metamodels, modeling environments, and transformations
- clearer relationship between MDA and service-oriented architecture (SOA) as a response to a growing software development trend
- further evolution and clarification of OMG specifications and initiatives

4.3 Acquirer Role-Based Perspective

In this perspective, the acquirer role is concerned with the ability to specify, in a contract, products or services that can be used to create, maintain, and field operational systems. Some current systems take an MDA approach, notably Single Integrated Air Picture (SIAP) and Joint Strike Fighter (JSF) [Jacobs 04].

Of special interest is the issue of what it means to acquire a model. For example, a contract could be awarded for development of a model, and subsequent implementations of that model could be separately awarded. One concern in that example would be the ownership of intellectual property rights. There are also acquisition strategy questions such as maturity and

[4] *Round-trip* refers to the capability to generate code from models and models from code, making it possible to make changes to generated code and then have those changes automatically reflected in the model.

risk of the technology, as well as the ability to integrate the model with other models that were possibly obtained through different contract vehicles.

The use of MDA is reminiscent of the transition to an object-oriented approach or to the use of computer-aided software engineering (CASE) tools. As with those techniques, there are concerns regarding cost, training, and the maturity of the technology with MDA. Furthermore, it is necessary to consider re-architecting legacy systems to foster integration with the capabilities provided by MDA technologies [Jacobs 04]. Thus, the transition to the use of an MDA approach can have broad effects on both an acquisition organization and a development organization.

Some of the perceived benefits to the acquirer regarding MDA include the following:

- It is easier to port software to a new computing environment or to defer the selection of hardware, middleware, and the operating system.
- The comprehensibility provided through the use of a model can foster competition in maintenance by soliciting a separate contract for maintenance, thereby reducing the life-cycle cost.

Some particular comments regarding the generator capability include the following:

- Some tools are able to generate code for popular operating systems, such as POSIX, VxWorks, or Win32. Tools can also provide extensibility to account for distribution middleware, such as common object request broker architecture (CORBA) or Java message service (JMS).
- The capability provided by some tools is rather basic. As a result, the acquirer must obtain code generation capabilities that are extensively tailored beyond those used for development of the PIM. This need for specialization reflects the immaturity of some code generators.
- There are challenges in porting the dynamic and nonfunctional aspects of PIMs from one tool to another.

4.4 Developer Role-Based Perspective

The most effective aspects of MDA in practice are the improved semantics for UML. There is also potential for the Systems Modeling Language (SysML) to solve a current problem with most UML-based tools—limited traceability between requirements and implementation.

From a developer perspective, there is potential for MDA in application and service development, but there should be caution: The current market hype may have overpromised what MDA can deliver.

From the developer's experience, there are aspects of MDA that are not as effective as might otherwise have been expected:

- OMG has done a good job of selling MDA, but not its interpretation, which has caused a proliferation of tools with their own interpretation of MDA, as well as multiple user expectations about what an MDA-based tool should do.
- Model transformation tools are slow to market.
- There are not enough automatic validation test generators.
- There is much confusion regarding the definition of computer-independent model (CIM), PIM, and PSM.
- Current tools generate code for a limited number of target platforms.
- Diagram interchange is difficult between tools from different vendors, let alone model interchange. Perhaps if MDA would have proposed MOF and not UML as the modeling tool, this difficulty would not occur.
- There is little tool support for domain-specific modeling.
- The MDA-based tool market is "business as usual" where many UML tools are being marketed as MDA tools.

There are also aspects that MDA does not address, **but that some community should address**, to make the use of MDA more effective.

- **tool interoperability**
  Even though the available tools claim to offer import and export capabilities, they provide only limited functionality beyond the exchange of diagrams that can be just a small portion of the complete model.
- **integration with complex analysis tools**
  It is not easy to make complex analysis tools work with UML-based tools to evaluate aspects such as schedulability and security or to do a dependency analysis.
- **theoretically sound operational semantics**
  There is considerable effort underway in OMG to develop a sound semantics of executable elements of UML. Progress has been slow, and the gains have been hard won. But these gains address only functional concerns—and only in an abstract (platform-independent) computational setting.
- **composition of models**
  Models must be transformed for multiple purposes—not just to generate code, but also to generate input to various analysis tools. Each such transformation must be backed by sound semantics. If multiple transformations are to be valid simultaneously (e.g., performance, functional, and security views), their semantics must be coherent.

Current MDA tools also (in general) lack capabilities to model and reason about nonfunctional attributes such as performance, security, and reliability—a limitation that is further explained in Section 5.5. In one example discussed in the workshop, the developer is

using the Vanderbilt Generic Modeling Environment (GME)[5] to model nonfunctional aspects such as timing and schedulability, deployment, and memory allocation. Basically, the developer has implemented the equivalent of a UML profile as GME metamodels. This solution works reasonably well, but there is no real support for transforming UML representations to specific analysis models. Therefore, this analysis occurs outside of the MDA-based tool.

Two recent experiences with code generators were described. In the first experience, a code generator was employed more as an "advanced build tool." Class and state diagrams were used to generate code to a real-time operating system (RTOS)/middleware isolation layer. C++ snippets were then assembled into compilation components. This solution was workable, but difficult to characterize as MDA-like. In the second experience, an MDA-based tool was used. The benefits of using that tool were the (1) easy access to model metadata for validation checks and documentation and (2) freedom it gave developers to focus on the problem space rather than the runtime mechanics.

Some lessons learned from the developer's experiences with code generators are

- Ensure that the tool is well suited to your problem domain. The tool used a very limited abstract language to represent actions that were then translated to code. For the developers, the language was not adequate for the complex mathematics required in some algorithms.
- Ensure that the tool vendor has a track record in your problem domain. For example, if the solution to the problem requires distribution and threading, the vendor should have experience using the tool in similar projects.
- Resist the urge to raise the abstraction level too high. The consequence of choosing the wrong level of abstraction is that the transformations may become unnecessarily complex.
- Ensure that the tool suite can scale up to industrial-size development teams. Some tools work well in a stand-alone environment but do not adapt well to distributed development and deployment environments.

MDA development is evolving and will evolve further—based on factors such as SOA, business process modeling, and model-driven development in general—into a yet unknown model architecture that may have its roots in OMG's MDA but be quite different. This path of development will require a paradigm shift that gives business users the power to define their processes and expected performance. By contrast, in current practice IT and business users seem to go in different directions. By empowering users, IT will be the "behind the scenes" provider of automated code generation that is platform independent and can be ported to specific platforms with ease.

[5] For more information, see http://www.isis.vanderbilt.edu/projects/gme/.

4.5 Researcher Role-Based Perspective

Generative programming is not a new idea: program generation has long been an intrinsic element of software development, even before the advent of FORTRAN. Indeed, the practice of software engineering would be inconceivable without technology to transform input specification to output object automatically. In this context, compiler technology (especially for conventional programming languages) is only the most mature and widely used, and therefore most familiar, example in this class of technology.

The search for more effective specification languages and processors continues, as seen by the long history of research in specification languages that pushes beyond the boundaries of conventional programming languages—including (but certainly not limited to) wide-spectrum languages, application-specific languages,[6] "4th-generation" languages, module-interconnection languages, architecture description languages, and composition languages.

It is difficult to assess the impact of this earlier research, at least when judged from its influence on the day-to-day activities of software developers. On the one hand, modern programming languages strongly reflect improved formal foundations, especially in type systems and module systems. On the other hand, software developers have stubbornly resisted any substantial shift from their fundamental reliance on imperative programming to higher-level specifications as a primary vehicle for implementation.

It is possible that changing factors in the research environments—including economic incentives, hardware capabilities, and social needs—may make some previously explored approaches more attractive. It is also possible that new forms of specification may be developed to address new classes of engineering problems. There is likely to be some degree of reinvention. Of course, it is not easy to draw the line between useful recapitulation and not-so-useful reinvention, and perhaps no line need be drawn.

What is indisputable is that the science, technology, and art of language design compose a significant body of knowledge. Figure 2 offers a research perspective on the evolution of program generation technology. This view is not offered as definitive or comprehensive, but instead as a reflection of *du jour* sensibilities about the subject. It adopts aspect-orientation as its organizing precept, implying an evolution from conventional to aspect-oriented programming and from aspect-oriented programming to model-based *something*—where the lack of an explicit compile step suggests a *something* that is qualitatively different from the conventional meaning of *programming*.

[6] *Application-specific languages* are also known as domain-specific or problem-oriented languages.

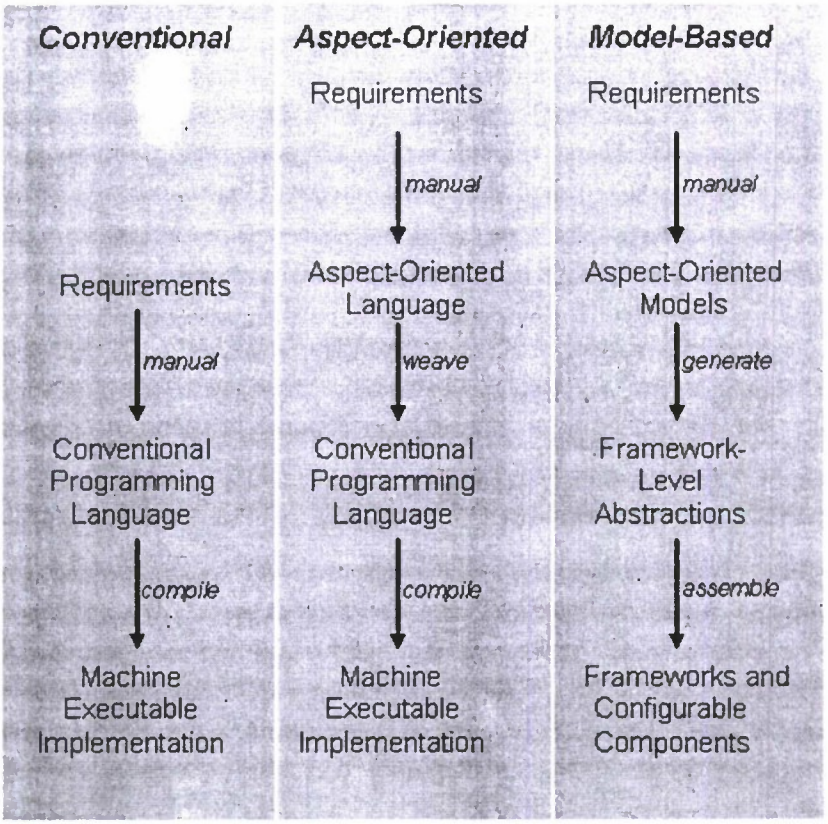

Figure 2: Three Contemporary Forms of Program Generation

MDA provides one interpretation of this *something*. Although it is not overtly "aspect-oriented," MDA, in its prescribed notations, does address distinct specification concerns (aspects). MDA also emphasizes metamodeling and imposes an organizational scheme (CIM, PIM, and PSM) to rationalize integration and transformation of concerns (weaving).

For a user, the ability to apply mature tools will lead to benefits expected of an MDA approach. Perhaps MDA, and model-generation tools in general, might exploit the technology to provide model-driven product lines or a model-based tool integration framework. Thus, although difficult problems remain in the application of MDA to large-scale systems, the experience provided by the research community will help industry to realize the expected benefits of MDA and other model-driven approaches to software development.

5 Thematic Perspectives

In the workshop, following presentations of the role-based perspectives, the group sought to identify themes that were relevant. These thematic perspectives are described below.

5.1 Understanding MDA

There is a lack of agreement concerning the term MDA.

It was recognized that the term *MDA* is used in several contexts, often simultaneously, and perhaps with different meaning! The term has been used to denote:

- a **philosophy** of development, stressing the use of models
- **standards** sponsored by the OMG that support a specific approach
- **products** developed by a vendor (that could relate to either of the above points)

MDA is sometimes viewed as a collection of standards, skills, tools, and processes that implement a particular approach to software development. It was pointed out, however, that there are no MDA tools *per se*. There are tools that implement a particular concept associated with MDA. In this sense, the concern of a philosophical approach is important to visionaries in the business communities. There are also implications regarding how to accomplish the practice of software engineering amid competing perspectives of the same term. Unfortunately, the lack of understanding around MDA percolates through various communities and causes further confusion.

5.2 Role of MDA in the Software Life Cycle

The role of MDA in the software life cycle could be greater, but current tools that support MDA do not make it easy.

Most current tools that support MDA focus on its code generation aspects: Models are developed using UML, and code is generated from these models. Even though this contribution is important and can potentially reduce development cycles, MDA could have a large impact on many other elements of the life cycle as well. For example, consider areas of the software life cycle that developers do not want to deal with, such as documentation, test cases, quality assurance, configuration management, and generation of "glue code." Given that a tool supporting MDA use throughout the software life cycle will probably never

become a reality,[7] there will be a need for more interoperability between tools at the model level.

Other tool features that would increase the role of MDA in the software life cycle include

- code generation for a greater variety of platforms
- debugging at the model level
- configuration management at the model level as opposed to the "XMI level"
- more modeling support for "upper stream" activities that provide traceability all the way to the generated code (This traceability would allow one to understand, for example, the impact of change to a business rule.)
- easier integration with analysis and validation tools

But more than anything else, there needs to be greater guidance in the use of MDA. How do you use MDA correctly? What does good modeling look like? What are design best practices? The answers to these questions are fundamental to extending the use of MDA further in the software life cycle.

5.3 Role of Transformations and Generator Technologies

The role of transformation and generator technologies is crucial to MDA.

A key aspect in the process of applying MDA is the transformation of a PIM to a PSM. The transformation is achieved through the use of some generator technology, the most common of which turns the model specification into executable code for a particular platform.

There are a number of aspects to consider regarding the generator technology. For example, an overriding issue is the amount of support that should be present in the model for nonfunctional attributes, as opposed to their specification in the context of a model's implementation.

It must also be recognized that generators can be developed for many purposes. For example, one could use output from a generator as input to a tool that performs an analysis of a model (independent of any implementation). Or, one could have a generator that creates test scenarios for a model. Perhaps, such generators can also be ported to the implementation context to obtain consistency between the PIM and PSM for various types of analysis.

A transformation, expressed through some generator technology, must address many aspects. Some are localized to the context of the domain defined by the model. For example, how does one demonstrate the correctness of a particular transformation? However, other considerations also apply. Most notably, these other aspects relate to interoperability concerns. For example, the integration with results produced by legacy systems or other

[7] For a pattern that MDA tool development might follow, we can look at the CASE experience.

model approaches is of importance—in particular, the ability to achieve semantic interoperability. Interoperability also becomes significant when one seeks interoperability among tools for possibly different purposes where different vendors have provided the tools.

Another constellation of issues arises when considering interoperation of language semantics, as opposed to interoperation of system semantics. Defining language semantics is difficult enough when there is only one intended target transformation. When there are multiple transformations (i.e., transform one model into many views), multiple semantic views must be mutually consistent. Questions of shared observational semantics and modular semantics, to name just two, pose challenges to state-of-the-research language specification technology.

In general, the efficiency of using MDA-based tools for code generation will depend on the amount of the programmer's effort that is automated by the tool. The level of automation is directly proportional to the sophistication of the transformations and to the amount of information that is present in the models and used by the transformations. As a result, transformations and generators will have to evolve beyond generating infrastructure code and code "skeletons," which most current MDA-based tools accomplish.

5.4 Present State of MDA Capabilities

The real-world application of MDA is still limited to certain problem domains.

MDA works well in business applications, which provide a well-understood problem domain dominated by functional requirements and standard platforms. MDA also works well in data-centric applications, because the underlying data models are very structured. For other types of problem domains, however, MDA has not performed as well. Even though there have been case studies on the use of MDA in real-time systems and other domains, these are situations that require large amounts of tailoring and vendor support—requirements that seem contradictory to the MDA philosophy.

At a different level of abstraction, MDA deals well with functional properties; it is less capable of dealing with nonfunctional attributes. There is debate about whether the specification of nonfunctional attributes is within the scope of MDA. Regardless of the resolution of that debate, there is a need for semantics that will allow the specification of nonfunctional attributes. This topic is further discussed in the next section.

5.5 Nonfunctional Attributes in a Model Context

There exists a need to represent, analyze, and verify nonfunctional attributes using models.

The term *nonfunctional attributes* refers to characteristics such as performance, reliability, maintainability, availability, and interoperability. The nature of this thematic perspective is directly related to the MDA approach.

In the context of a PIM, the emphasis is on functional attributes of the system being modeled. For example, suppose one has developed a PIM for some application. It may be possible to simulate the PIM, but that simulation would be in the context of a single process on a single processor. At that level of abstraction, it is not possible to address such characteristics as timing properties. This example illustrates, simply, the concept that a PIM abstracts implementation-level details. Unfortunately, there are cases where such details are important.

The transformation from PIM to PSM is determined, at least in part, by the ability to satisfy nonfunctional attributes. Consider the case where there is a requirement about timing (such as the amount of time required to transmit a message):

- A timing requirement could be **represented** in the context of a model. Such an expression would be one of intent, in effect a somewhat refined statement of the requirement.
- **Analysis** of a timing requirement in the context of a PIM could be performed **qualitatively**. For example, one may assess the temporal ordering of a sequence of events.
- **Verification** of the timing requirement requires consideration of the execution context of the code-generated form of the model. Thus, verification is not possible until one enters the context of platform-specific considerations. This verification includes **quantitative** analysis of the requirement as well. For example, the value of a temporal interval is influenced by implementation concerns, notably the operating system.

This separation of perspectives (model versus implementation) for nonfunctional attributes has limited the utility of MDA in certain environments. More can be done to abstract such discussion to the context of a model; in the end, however, it is the implementation context that is most important.

5.6 Validation

MDA could play a larger role in system validation.

In an ideal model-driven development scenario, there are a number of models of a system, interrelated in a way that provides traceability (e.g., from use cases to code). In this ideal scenario, it should be possible to

- perform model checking or generate test cases to determine if all requirements are being satisfied
- generate load tests
- execute incremental validation so that the whole system does not have to be revalidated when a part of it changes
- carry out analysis on the impact of changes in requirements or design
- validate consistency of life-cycle artifacts

Because some of these validations are performed on models rather than on code, there is the potential to include a broader range of people in the validation process and introduce that process earlier in the development life cycle.

Unfortunately, this scenario is not currently viable because most tools do not provide the necessary traceability between code and "upper stream" activities, as mentioned in Section 5.2. Some tools are starting to support validation by providing some form of automated test case generation and coverage testing, but their functionality in those areas is still very limited.

5.7 Domain-Specific Languages

Domain-specific languages (DSLs) provide a way of incorporating domain elements into a model; but creating a DSL, and doing it right, is not easy.

The skilled use of object-oriented modeling produces domain models that are then composed, programmatically, into systems. Adding metamodeling to the mix seems just a small technical step. In fact, though, these additions involve the MDA developer in language design, something that requires considerable technical skill—if it is to be done soundly. This is especially true if we also include model transformation in the mix, which invites comparison with compilation and (model-theoretic) interpretation.

The reason there are not many DSLs overtly incorporated into modeling tools is probably that they are difficult to design, implement, and maintain. For example, the proper design of a DSL requires the use of domain engineering to

- determine the right scope for the language
- determine the elements of the domain that reside within this scope
- describe each element and its relationships with other elements

Even if these technical challenges can be skillfully met, a proliferation of DSLs could lead to a "Tower of Babel" that in turn would further decrease model-exchange capabilities between modeling tools and with analysis tools, as mentioned in Section 4.4.

A potential solution to this problem is to have a metamodel for DSLs. There is currently a great amount of discussion on the creation of a UML Profile for DSLs, which would be a step in this direction. On the other hand, this entire subject may be more suitable for research than for standards development.

5.8 SEI Role

An additional perspective discussed was the role the SEI could play in the technology associated with MDA. The topics discussed in this area include

- **education**
  Throughout the workshop, it was recognized that there is confusion about how the term *MDA* is used. The SEI could help to "separate fact from fiction." It was pointed out, for example, that MDA in itself is not a solution to the interoperability problem. The means by which the SEI can clarify use of the term might include an integrated glossary and courses intended for executives.
- **experimentation**
  There would be value in performing experiments that would provide knowledge to the community.
- **transition**
  A main function of the SEI is to transition technology, and several ideas were brought forward. One was for the SEI to interact with DoD programs using MDA, as well as with vendors and researchers. Bringing these groups together would help develop a shared understanding of problems and approaches. It was also suggested that the SEI could expand its participation with OMG.
- **research**
  A research agenda relating to MDA technologies would be of value. As part of this subject, participants perceived that consideration of technology readiness levels (TRLs) and a technology maturity model would be useful. There are also research questions that apply to the acquisition community (e.g., what does it mean to acquire a model?)

6 Summary

This technical note has addressed various aspects related to MDA, which is suggested as a means to develop and maintain large systems. Along with the touted advantages of using MDA, there are a number of concerns. The workshop attendees represented perspectives of the standards body, vendor, acquisition, development, and research communities. In addition to considering these perspectives, workshop attendees developed several integrated perspectives.

Three conclusions stand out from this workshop:

1. OMG's MDA is an approach to software development and not just a set of standards. The standards exist to support the realization of this approach, especially as they relate to the transformations of a PIM to one or more PSMs and the capability for tools to support these transformations.
2. MDA tools available today exhibit useful capabilities for generating software from models. Nonetheless, the technical challenges posed by the MDA vision push beyond the reach of what can be achieved with today's language specification technology, especially with respect to nonfunctional semantics (behavior). There is great risk that consumer expectations will not be satisfied.
3. The question of acquisition of a model is problematic. It runs the gamut from high-level acquisition strategy to the language of a contract. To the extent that MDA represents an abstraction of a PSM to a PIM, there may be a corresponding abstraction required from the perspective of an acquisition.

In summary, while the principles underlying MDA are sound, there remain issues that must be solved for MDA to realize its full potential.

References

URLs are valid as of the publication date of this document.

[ATL 04] Office of the Under Secretary of Defense For Acquisition, Technology, and Logistics. *Report of the Defense Science Board Task Force on Integrated Fire Support in the Battlespace.* Washington, DC, October 2004. http://www.acq.osd.mil/dsb/reports/2004-10-Integrated_Fire_Support.pdf.

[Jacobs 04] Jacobs, Robert W. "Model-Driven Development of Command and Control Capabilities for Joint and Coalition Warfare," *2004 Command and Control Research and Technology Symposium.* San Diego, CA, June 15–17, 2004. http://www.dodccrp.org/events/2004_CCRTS/CD/papers/260.pdf.

[Lewis 04] Lewis, Grace A. & Wrage, Lutz. *Approaches to Constructive Interoperability* (CMU/SEI-2004-TR-020, ADA431067). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tr020.html.

[OMG 03] Object Management Group. *MDA Guide Version 1.0.1.* http://www.omg.org/docs/omg/03-06-01.pdf (2003).

[OMG 06] Object Management Group. *Executive Overview: Model-Driven Architecture.* http://www.omg.org/mda/executive_overview.htm (2006).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	August 2006	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Workshop on Program Generation and Model-Driven Architecture	FA8721-05-C-0003

6. AUTHOR(S)

Grace A. Lewis, B. Craig Meyers, Kurt C. Wallnau

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2006-TN-031

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This technical note summarizes the results of a workshop held on June 2, 2006, at the Software Engineering Institute in Pittsburgh, Pennsylvania (USA). The workshop explored business and technical aspects of program generation in the context of the Object Management Group's model-driven architecture development approach. The workshop was structured around consideration of the perspectives of five different communities: standards body, vendor, acquisition, development, and research. This note recapitulates these individual perspectives and highlights important themes.

14. SUBJECT TERMS	15. NUMBER OF PAGES
model-driven architecture, MDA, software architecture, architecture-based design, model-based design, architecture model	36

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL